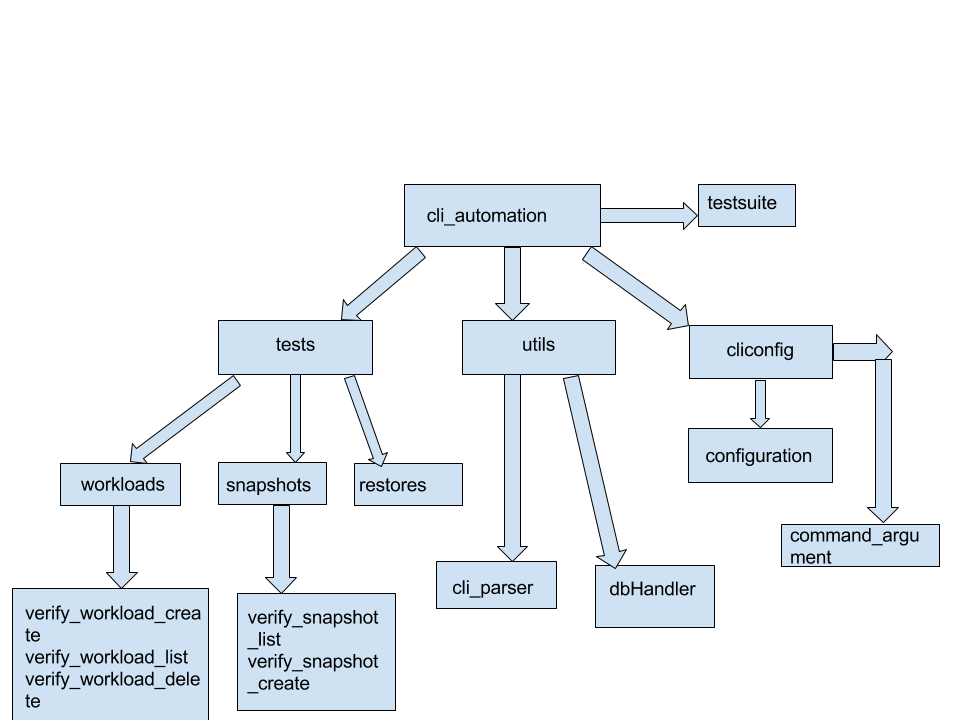
**CLI Automation Framework Structure**

CLI Automation Framework has been designed for automating the CLI of TrilioVault. It comprises of following architecture:



The framework is structured as shown in the diagram.

The package cli\_automation consists of following components:

1. **CliConfig:** This package is primarily built for configuration purposes.
2. Configuration.py contains static names for workload, snapshot and restore. Any workloads, snapshots or restores created on tvault take the name given in configuration.py file.
3. Command\_argument\_string.py contains different commands stored in variables. If any command gets changed, it should be changed in command\_argument\_string.py file so that testscript file need not be changed.
4. **Utils:** This package contains all the modules used for building libraries.
5. Cli\_parser.py contains the methods for calling subprocess which in turn call the tvault command and return command exit code or the command output. One method is responsible for returning the exit code of command and the other returns the output of command.
6. dbHandler.py contains the code for connecting to tvault database. It uses the credentials and hostname given in configuration.py to connect to tvault mysql database.
7. Query\_data.py contains the functions for fetching data from tvault database for verification.
8. HTMLTestRunner.py contains the code for generating reports that are compatible with python unittest framework.
9. **Tests:** This package contains test scripts related to functionalities like workloads, snapshots and restores. Each functionality is kept as a separate package which contains test script files for respective functionality commands. So tests package comprises of following sample structure:
10. workloads
11. verify\_workload\_create.py
12. verify\_workload\_list.py
13. snapshots
14. verify\_snapshot\_create.py
15. verify\_snapshot\_list.py
16. restores
17. verify\_oneclick\_restore.py
18. verify\_restore\_list.py
19. **testsuite.py:** This file is for running all the tests together. It creates a testsuite using python unittest framework and runs that suite. After the run is complete, it creates a HTML report using HTMLTestRunner.py which gives detailed report of all the tests run and its final status (Pass/Fail)

**Pre-requisites for executing the framework**

1. Clone the github repo <https://github.com/trilioData/tempest.git> in /opt/stack location on the machine where execution is to be triggered.
2. Install mysql connector for python
3. Install python workloadmgr client and all other python openstack clients like nova client, glance client etc.
4. Change the tvault ip under cliconfig/configuration.py file
5. Source adminrc file for openstack

**Adding tests to the framework**

Follow following steps to add new tests to the framework:

1. Create a new test script under relevant packages e.g workload command test script under workloads package, snapshot command test script under snapshots package and so on.
2. Use the unittest framework structure like the one already being used in other tests.
3. Add any database verification methods to query\_data.py file under utils package.
4. Add the test to the test suite in testsuite.py file.

**Executing the CLI automation framework**

Run the testsuite.py using the command **python testsuite.py** to run all the tests and after execution, HTML reports can be viewed under Reports directory. Standalone tests can also be run for verification purposes using python <test\_name.py> but that will not generate HTML report.